ACM321 OBJECT ORIENTED PROGRAMMING LAB NOTES

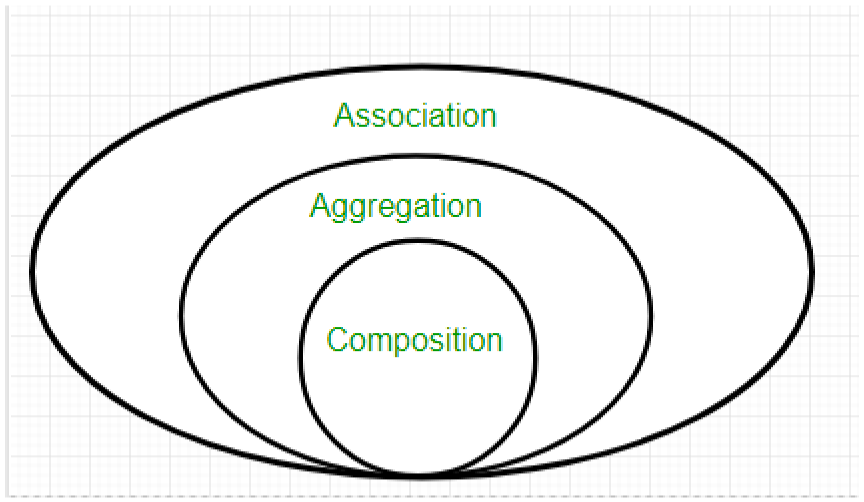
Class Relationship

OUTLINE

- Dependency A uses B

- Association
- Aggregation A has-a B
- Composition B part-of A

- Inheritance A is-a B (LATER ON THE COURSE)



Dependency:

Dependency Types:

We have different type of dependencies. Each type leads to more or less flexibility in the code.

> Class Dependencies

Class dependencies are dependencies on classes.

For instance, the method in the below code box takes a String as parameter. Therefore, it depends on the String class.

```
//class dependency - it depends on String class
public byte[] readFileContents(String fileName){
     //some code
}
```

> Interface Dependencies

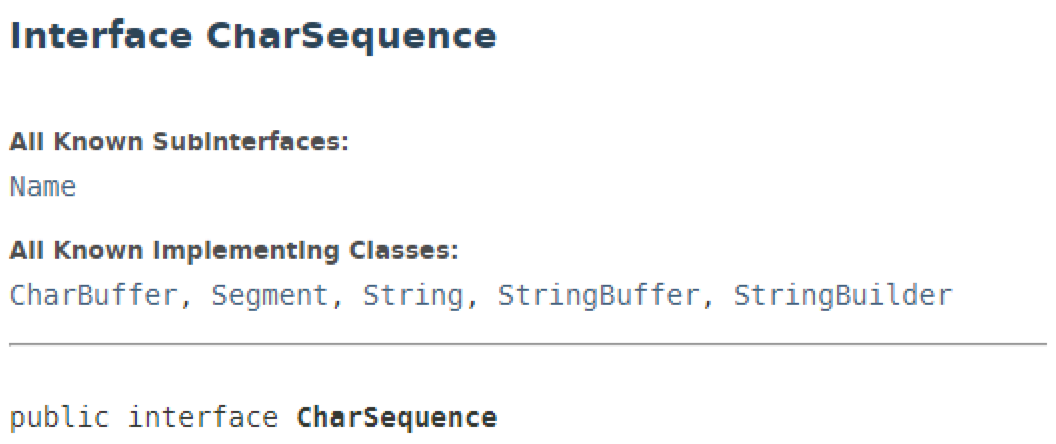
Interface dependencies are dependencies on interfaces.

The method in the below code box takes a CharSequence as parameter.

Both **CharBuffer**, **String**, **StringBuffer**, and **StringBuilder** implements the **CharSequence** interface, so instances of any of these classes can be used as parameter to the method.

```
//interface dependency - it depends on CharSequence interface
public byte[] readFileContents(CharSequence fileName){
     //open the file and return the contents as a byte array.
}
```

Document of CharSequence:
https://docs.oracle.com/javase/8/docs/api/java/lang/CharSequence.html


Interface CharSequence

All Known Subinterfaces:
Name

All Known Implementing Classes:
CharBuffer, Segment, String, StringBuffer, StringBuilder

public interface CharSequence

> Method Dependencies

One class dependent to the method of another class.

For example, think about the Player and Dice object. Dice neither a part of a Player. Player just need its functionality, in that case Roll() method. So that Player depends on Dice class via Roll() method.

```java
class Dice{
	public int Roll() {
		return 6;
	}
}

class Player{
	private String player_name;

	Player(String name){
		this.player_name = name;
	}

	//There is a dependency between the Player class and Roll method of Die class.
	public int moveTurn(Dice die1) {
		//some code
		int result = die1.Roll();
		//some code...
		return result;
	}
}

public class MethodDependency {

	public static void main(String[] args) {
		Dice die = new Dice();
		Player player1 = new Player("Mehmet Ali");
		System.out.println(player1.moveTurn(die));
	}

}
```

Association

Association is reference based relationship between two classes. Here a class A holds a class level reference to class B.

```
// class Car
class Car {
	private String car_brand;
	private Engine engine;

	public Car(String brand, Engine engine) {
		this.car_brand = brand;
		this.engine = engine;
	}

	public String getCarBrand() {
		return this.car_brand;
	}

	public String getEngineModel() {
		return this.engine.getModel();
	}
}

//There is a relationship between Engine Class and Car Class

//Engine Class
class Engine {
	private String model;
	private String material;
	private String fuel_type;

	public Engine(String model, String material, String fuel_type) {
		this.model = model;
		this.material = material;
		this.fuel_type = fuel_type;
	}

	public String getModel() {
		return this.model;
	}
}

public class AssociationDemo {
	public static void main(String[] args) {
		Engine enginev8 = new Engine("BMW M62", "Aliminium",
"Petrol");
		Car car_bmw = new Car("BMW",enginev8);

		System.out.println(car_bmw.getEngineModel() + " is the
engine model!");

	}

}
```

Aggregation

Aggregation is same as association and is often seen as redundant relationship. A common perception is that aggregation represents one-to-many / many-to-many / part-whole relationships.

AggregationDemo.java

```
//We import the java.util.List because we are going to use when we
define List<Student>.
import java.util.List;
import java.util.ArrayList;

class Student{
	private String full_name;

	Student(String fullName) {
		this.full_name = fullName;
	}

	public String getFullName() {
		return this.full_name;
	}
}
class Classroom{
	private String class_name;
	private List<Student> students;

	Classroom(String class_name, List<Student> student_list) {
		this.class_name = class_name;
		this.students = student_list;
	}
	public String getClassName() {
		return this.class_name;
	}
	
	public List<Student> getStudentsinClass(){
		return this.students;
	}
	
	//we can set new students
	public void setStudent(List<Student> students) {
		this.students = students;
	}
}

public class AggregationDemo {

	public static void main(String[] args) {

		List<Student> acm_students = new ArrayList<Student>();

		Student person1 = new Student("Mehmet Ali Özer");
		Student person2 = new Student("Aryen Hemvatan");
		Student person3 = new Student("Chloe Stewart");
		Student person4 = new Student("Lauren Hanover");

		acm_students.add(person1);
		acm_students.add(person2);
		acm_students.add(person3);
		acm_students.add(person4);

		Classroom classACM1 = new Classroom("ACM321 OOP", 
acm_students);

		List<Student> registered_student = 
classACM1.getStudentsinClass();
		for(int i=0; i < registered_student.size(); i++) {
			
	System.out.println(registered_student.get(i).getFullName());
		}

	}
}
```

Composition

Composition relates to instance creational responsibility. When class B is composed by class A, class A instance owns the creation or controls lifetime of instance of class B. Needless to say when class instance A instance is destructed (garbage collected), class B instance would meet the same fate. Composition is usually indicated by line connecting two classes with addition of a solid diamond at end of the class who owns the creational responsibility. It's also a perceived wrong notion that composition is implemented as nested classes. Composition binds lifetime of a specific instance for a given class, while class itself may be accessible by other parts of the system.

```java
// class Car
class Car {
	private String car_brand;
	private Engine engine;

	public Car(String brand, String model, String material, String 
fuel_type) {
		this.car_brand = brand;
		this.engine = new Engine(model, material, fuel_type);
	}

	public String getCarBrand() {
		return this.car_brand;
	}

	public String getEngineModel() {
		return this.engine.getModel();
	}
}

//There is a relationship between Engine Class and Car Class

//Engine Class
class Engine {
	private String model;
	private String material;
	private String fuel_type;

	public Engine(String model, String material, String fuel_type) {
		this.model = model;
```

```
        this.material = material;
        this.fuel_type = fuel_type;
    }

    public String getModel() {
        return this.model;
    }
}

public class CompositionDemo {

    public static void main(String[] args) {
        Car car_bmw = new Car("BMW", "BMW M62", "Aliminium",
"Petrol");

        System.out.println(car_bmw.getEngineModel() + " is the
engine model!");
    }
}
```

Aggregation vs Composition

Dependency: Aggregation implies a relationship where the child can exist independently of the parent.

For example, Bank and Employee, delete the Bank and the Employee still exist, whereas Composition implies a relationship where the child cannot exist independent of the parent.

Example: Human and heart, heart don’t exist separate to a Human
Type of Relationship: Aggregation relation is “has-a” and composition is “part-of” relation.

Type of association: Composition is a strong Association whereas Aggregation is a weak Association

Summary Comparison Table

Let's check below table for association, aggregation and composition brief summary:

	Association	**Aggregation**	**Composition**
Related to Association		Special type of Association.	Special type of Aggregation
	8	Weak association	Strong association
Relation		**Has-A**	**Owns (part of ..)**
		one object is the owner of another object.	one object is contained in another object
Owner	No owner	Single owner	Single owner
Life-cycle	own life-cycle	own life-cycle	owner life-cycle
Child object	independent	belong to single parent	belong to single parent

Dökümanlar aşağıdaki kaynakların birleşimidir.

Ayrı ayrı okuyabilirsiniz.

https://www.geeksforgeeks.org/association-composition-aggregation-java/

https://nirajrules.wordpress.com/2011/07/15/association-vs-dependency-vs-aggregation-vs-composition/

https://www.dariawan.com/tutorials/java/association-aggregation-and-composition-in-java/